Unit 1:

- Vector
- Conditional Structure
- Loop
- Function
- Input and Output

Data Structure: Vector

- A vector is a set of same type of data: numeric, logic, text, etc.
- To create a vector, use:

variable “=“ “c” + “(“ + elements sepate with coma + “)”

A= c(1,5,3,4) [1]1 5 3 4	#numerical vector with 4 elements
B =c(T,F,T,F,T) [1] TRUE FALSE TRUE FALSE TRUE	#logical vector with 5 elements
D= c(“leganes”, “avila”, “getafe”) [1] leganes avila getafe	#vector with 3 text string

Data Structure: Vector

- To concatenate two vectors, you have to use: "c"

a=c(1,3,5) **b=c(2,4,6)**	
d=c(a,b)	d=c(b,a)
[1] 1 3 5 2 4 6	[1] 2 4 6 1 3 5

SEL-Promise

Software Engineering Lab

Data Structure: Vector

- To select elements of a vector, you have to use the position of the element in the vector:

variable “=“ vector_name + “[“ + position + ”]”

or

variable “=“ vector_name + “[“ + “c” + “(“ + position separate with comma + “)” + ”]”

>d=c(1,3,5,2,4,6)	
>d[2]	d[c(1,3,6)]
[1] 3	[1] 1,5,6

Data Structure: Vector

- Other example:

```
A=1
B=3
C=5
```

```
p=c(A,B,C)
[1] 1  3  5
```

```
D= p[B]
[1] 5
```

Data Structure: Vector

- To eliminate same element of a vector, you have to use:

variable “=“ vector_name + “[“ + “-” + position + ”]”

or

variable “=“ vector_name + “[“ + “- ” +“c ” + “(“ + position separate with comma + “)” + ”]”

>d=c(1,3,5,2,4,6)	
>d[-2]	d[-c(1,3,6)]
[1] 1 5 2 4 6	[1] 3 2 4

Data Structure: Vector

- Some functions with vectors

A=c(1,3,5,2,4,6)	
>sum(A) [1]21	Sum of vector elements
>min(A) [1]1	The least number of the vector
>max(A) [1]6	The greatest number of the vector
>length(A) [1]6	Numbers of vector elements
>range(A) [1]1 6	The least and greatest number of the vector
>mean(A) [1]3.5	The mean of vector elements
>sort(A) [1]1 2 3 4 5 6	The vector is ordered into ascending order

Data Structure: Vector

- You can name vector elements with the function: name()

```
>simpsons=c("Homer", "Marge, "Bart","Lisa","Maggie")

>names(simpsons)=c("dad", "mom", "son", "daughter 1", "daughter 2")

>simpsons
     dad       mom      son    daughter1   daughter2
  "Homer"   "Marge"  "Bart"     "Lisa"       "Maggie"
```

Data Structure: Vector

- To select with a logical condiction:

```
>d=c(1,3,5,2,4,6)
```

```
>d>3
[1]  FALSE  FALSE  TRUE  FALSE  TRUE  TRUE

#   if d is greater than 3
```

```
>d[d>3]
[1] 5 4 6

#   select numbers greatest than 3
```

Data Structure: Vector

- Logical operators:

```
>d=1:5

>d>1                                        #   if d is greater  than 1
[1] FALSE TRUE TRUE TRUE TRUE

>d < 5                                      #   if d is less than 5
[1]TRUE TRUE TRUE TRUE FALSE

>d>1 & d<5
[1] FALSE TRUE TRUE TRUE FALSE

#d greater than 1 AND d less than 5

>d>1 | d<5
[1] TRUE TRUE TRUE TRUE TRUE
#d greater than 1 OR d less than 5
```

Logical Operators

Value	OPERATOR	Value	Result
FALSE	AND	FALSE	FALSE
FALSE	AND	TRUE	FALSE
TRUE	AND	FALSE	FALSE
TRUE	AND	TRUE	TRUE

Value	OPERATOR	Value	Result
FALSE	OR	FALSE	FALSE
FALSE	OR	TRUE	TRUE
TRUE	OR	FALSE	TRUE
TRUE	OR	TRUE	TRUE

Data Structure: Vector

- More logical operators:

```
>d=1:5
```

```
>d==3                                    #   d as same as 3
[1] FALSE FALSE TRUE FALSE FALSE
```

```
>d !=3                                   #   d different to 3
[1]TRUE TRUE FALSE TRUE TRUE
```

```
>! (d==3)                                #no (x as same as 3)
[1]TRUE TRUE FALSE TRUE TRUE
```

Conditional Structure: if()

- It's a control statements.
- It allows you depending on whether a condition is met, perform different actions
- Syntax:

```
if (condition)
  {
     sentences A
  }
else
  {
    sentences B
  }
```

```
grade=5
if (grade>=5)
   print("pass")
else
   print ("fail")
```

```
[1]   pass
```

Conditional Structure: if()

- You can write nested conditional sentences.

- Syntax:

```
if (condition A)
   {
      sentences P
   }
else
 {
   if (condition B)
       {
         sentences M
       }
  else
       {
         sentences S
       }
 }
```

- "conditional A" and "condition B":
  -- return a logical value
  -- can be simple or multiple (AND, OR)

```
grade=9
if (grade<5)
{
    print("fail")
}
else
  {
   if((grade>=5) & (grade<7))
         print("pass")
     else
          print("with honors")
     }
```

```
[1]   with honors
```

Loop: for()

- When you need to repeat the same operations n times:

```
for (vble in list)
{
     sentences
}
```

- Example:

```
for(i in 1:5)
 {
    print(i)
 }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Loop: for()

- Exercise: To show elements of a vector (A: 1,3, 5, 4)

Loop: for()

- **Exercise**: To show elements of a vector

```
A=c(1,3,5,4)
for(i in 1:4)
  {
     print(A[i])
 }
```

```
[1] 1
[1] 3
[1] 5
[1] 4
```

If the vector is modified, what would happen with this solution?

Loop: for()

- Exercise: To show elements of a vector

```
A=c(1,3,5,4)
for(i in 1:4)
  {
     print(A[i])
 }
```

```
[1] 1
[1] 3
[1] 5
[1] 4
```

If the vector is modified, what would happen with this solution?

It doesn't work

Loop: for()

- **Example**: To show elements of a vector.

```
A=c(1,3,5,4)
for(i in 1:4)
 {
    print(A[i])
}
```

```
[1] 1
[1] 3
[1] 5
[1] 4
```

```
A=c(1,3,5,4)
for(i in 1:length(A))
 {
    print(A[i])
}
```

```
[1] 1
[1] 3
[1] 5
[1] 4
```

Function

- Syntax:

```
Name_function <- function(arg_1,arg_2,…,arg_n)
{
    sentences
    #return a value
}
```

- To return a value before the function is finished

```
return(variable/expression)
```

- To call a function

```
Name_function (expr_1, expr_2,…,expr_n)
```

Function

- Example:

```
> myfirstfunction<-function()
 {
     a=4
     b=5
     c=a+b
     return(c)
 }
```

```
> myfirstfunction()
[1] 9
```

A function to sum two values

Execution: you have to do the call to the function

Input

- User can enter a value in the console during execution using different functions:

```
>readline(prompt = "")

Example:
>colour=readline(prompt = "Write a colour: ")

--------The user will read

Write a colour:

----------User will write:  red

Write a colour: red

----------- the variable colour has the red value

> colour
[1] "red"
```

Input

scan (file = "", what = double(), nmax = -1, n = -1, sep = "", quote = if(identical(sep, "\n")) "" else "'\"", dec = ".", skip = 0, nlines = 0, na.strings = "NA", flush = FALSE, fill = FALSE, strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE, comment.char = "", allowEscapes = FALSE, fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

Example:

```
> colour=scan(, what=character(),2)
--------The user will read
1:
----------User will write:  red
1: red
-------- The user will read
> colour=scan(, what=character(),2)
1: red
2:
----------User will write:  blue
> colour=scan(, what=character(),2)
1: red
2:blue
Read 2 items

----------the variable colour has the following values:

> colour

[1] "red"  "blue"
```

Output

- Show data on screen during the execution of a program
  - print()
  - cat()

```
> cat("These are the main options.\n
 1.- Option 1\n
 2.- Option 2\n")
```

```
---Execution; the user will read:

These are the main options.
    1.- Option 1
    2.- Option 2
```

```
>print("Hello")
```

```
---Execution; the user will read:

Hello
```

SEL-Promise

Exercise: vector

- Create a vector with these elements: 2,17,15,7,11,3,8,19
- Calculate:
  - the maximum
  - the minimum
  - the length of the vector
  - the first element of the vector
  - the last element of the vector
  - the accumulated
  - the range
  - average
  - order from lowest to highest vector elements

Exercise: vector

- order from highest to lowest vector elements
- the square of each vector element
- the sum of the vector elements
- Is each element of the vector greater than 5?
- show each of the values of vector elements that is greater than 5.
- select the first three elements of the vector
- select the first, third and fourth element
- exclude the second, third and sixth element
- exclude the 4th element of the vector
- assign the value 6 to the third vector element
- assign the value 8 and 2 to the third and fifth vector element